Intel® Math Kernel Library
(Intel® MKL)

Intel® Math Kernel Library (Intel® MKL) Introduction

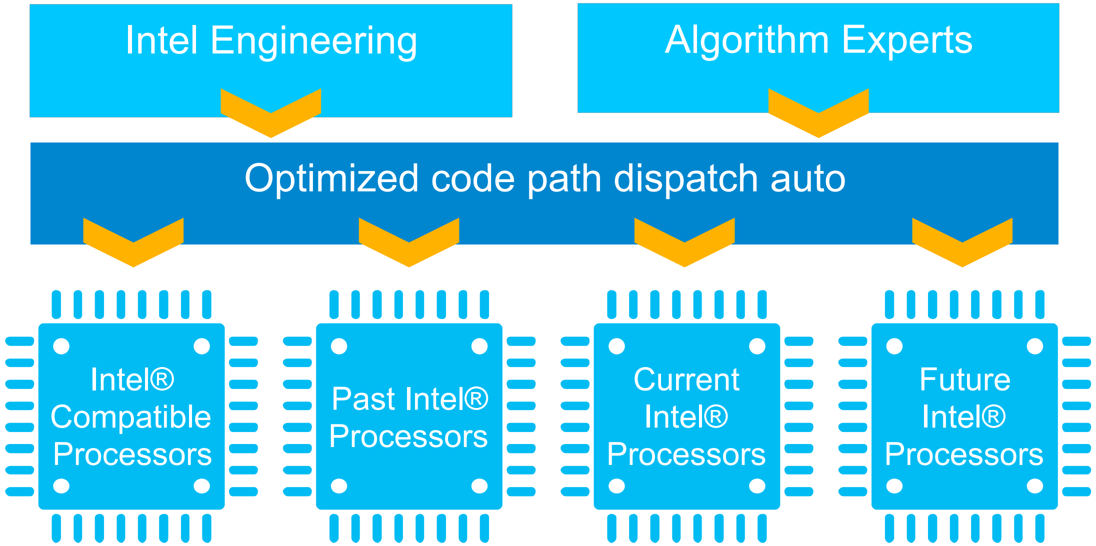
Highly optimized threaded math routines

- Performance, Performance, Performance!

Industry's leading math library

- Widely used in science, engineering, data processing

Tuned for Intel® processors – current and next generation



Evans Data Corporation EDC

EDC North America Development Survey 2016, Volume I

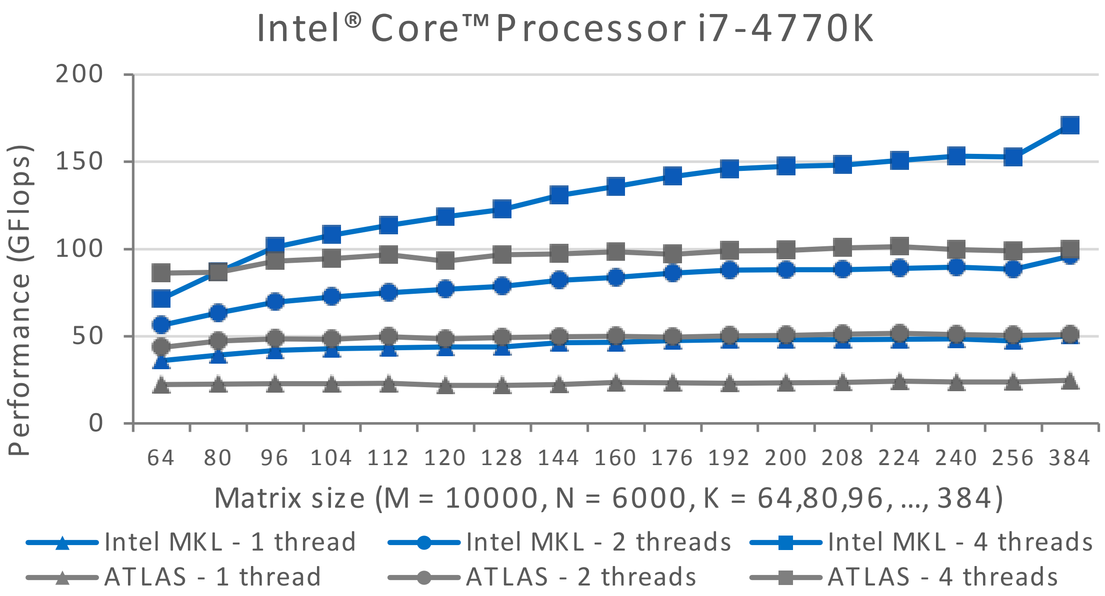
More math library users depend on MKL than any other library

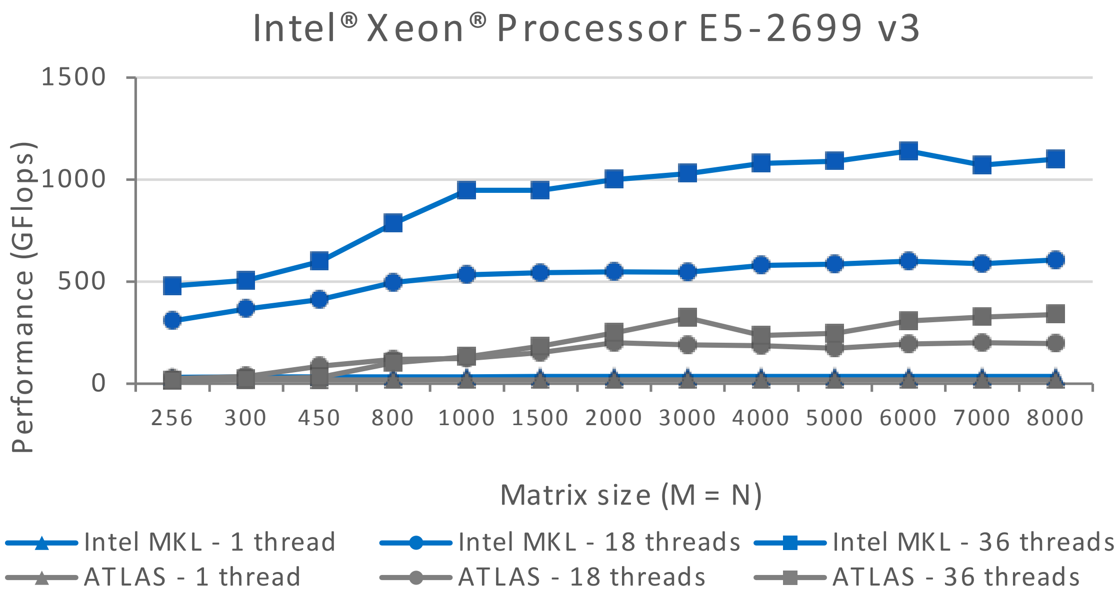
Be multiprocessor aware

- Cross-Platform Support
- Be vectorised , threaded, and distributed multiprocessor aware

Intel MKL unleashes the performance benefits of Intel architectures

DGEMM Performance Boost by using Intel® MKL vs. ATLAS*





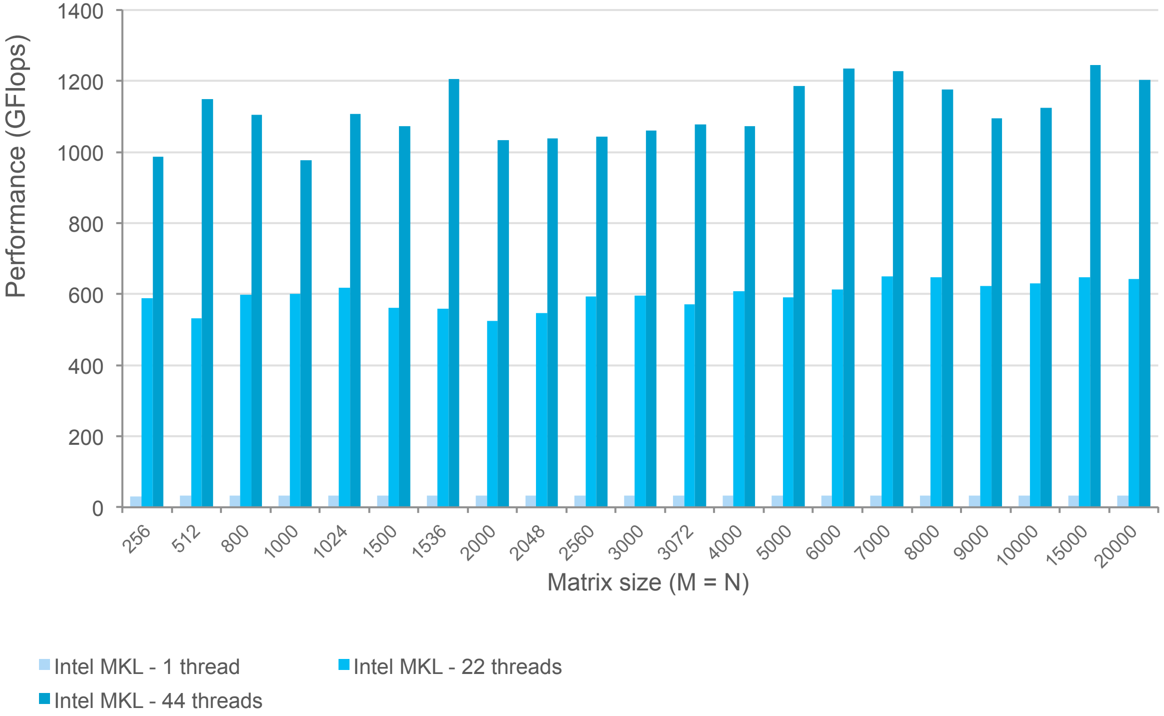
Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 11.3, ATLAS* 3.10.2; Hardware: Intel® Xeon® Processor E5-2699v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 64GB of RAM; Intel® Core™ Processor i7-4770K, Quad-core CPU (8MB LLC, 3.5GHz), 8GB of RAM; Operating System: RHEL 6.4 GA x86_64;

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. *Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

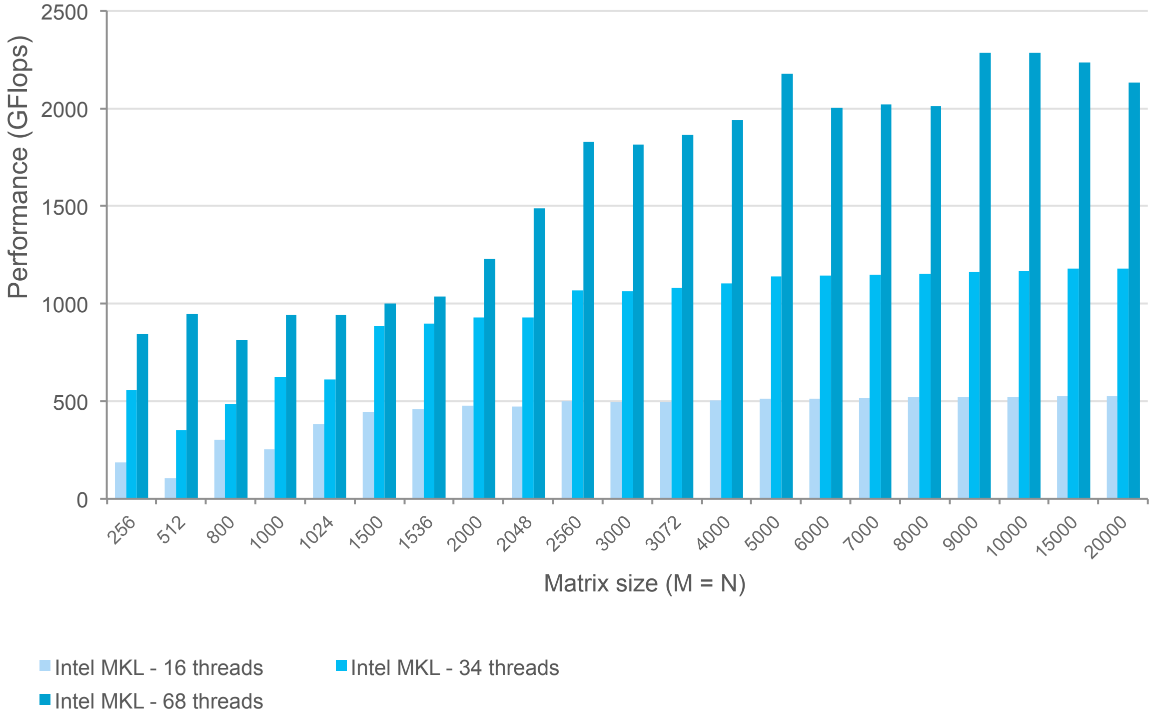
Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Intel MKL 2017 Performance

DGEMM Performance
On Intel® Xeon® Processor E5-2699 v4



DGEMM Performance
On Intel® Xeon Phi™ Processor 7250



Configuration Info - Versions: Intel® Math Kernel Library (Intel® MKL) 2017; Hardware:Hardware: Intel® Xeon® Processor E5-2699 v4, 2 Twenty-two-core CPU (55MB smart cache, 2.2GHz), 64GB of RAM; Intel® Xeon Phi™ Processor 7250, 68 cores (34MB L2 cache, 1.4GHz), 96 GB of DDR4 RAM and 16 GB MCDRAM; Operating System: RHEL 7.2 GA x86_64;

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Optimized Mathematical Building Blocks

Linear Algebra	Fast Fourier Transforms	Vector Math	Vector RNGs	Summary Statistics	Deep Neural Networks (DNN)
• BLAS • LAPACK • ScaLAPACK • Sparse BLAS • Sparse Solvers • Iterative • PARDISO* SMP & Cluster	• Multidimensional • FFTW interfaces • Cluster FFT	• Trigonometric • Hyperbolic • Exponential • Log • Power • Root	• Congruential • Wichmann-Hill • Mersenne Twister • Sobol • Neiderreiter • Non-deterministic	• Kurtosis • Variation coefficient • Order statistics • Min/max • Variance-covariance	• Convolution • Pooling • Normalization • ReLU • Softmax

BLAS – Basic Linear Algebra Subprograms

Defacto-standard APIs since the 1980s (Fortran 77)

- Level 1 – vector-vector operations
- Level 2 – matrix-vector operations
- Level 3 – matrix-matrix operations

Original BLAS available at
http://netlib.org/blas/

- Precisions: single, double, single complex, double complex

Operation	MKL Routine “D is for double”	Example	Computational complexity (work)
Vector Vector	D AXPY	$y = y + \alpha x$	$O(N)$
Matrix Vector	D GEMV	$y = \alpha Ax + \beta y$	$O(N^2)$
Matrix Matrix	D GEMM	$C = \alpha A * B + \beta C$	$O(N^3)$

LAPACK – Linear Algebra PACKage

Defacto-standard APIs since early 1990s

1000s of linear algebra functions

4 floating point precisions supported

Breadth of coverage:

- Matrix factorizations: the 3 Amigos – LU, Cholesky, QR
- Solving systems of linear equations
- Condition number estimates
- Singular value decomposition
- Symmetric and non-symmetric eigenvalue problems
- And much, much more

Original LAPACK is available at:
http://netlib.org/lapack/

Fast Fourier Transform (FFT)

Support multidimensional transforms

Multiple transforms on single call

Input/output strides supported

- Allow FFT of a part of image, padding for better performance, transform combined with transposition, facilitates development of mixed-language applications.

Integrated FFTW interfaces

- Source code of FFTW3 and FFTW2 wrappers in C/C++ and Fortran are provided.
- FFTW3 wrappers are also built into the library.

Vector Math Functions

Example: $y(i) = e^{x(i)}$ for $i = 1$ *to n*

- Arithmetic
  - add/sub/sqrt/ ...
- Exponential and log
  - exp/pow/log/log10
- Trigonometric and hyperbolic
  - sin/cos/sincos/tan(h)
  - asin/acos/atan(h)
- Rounding
  - ceil, floor, round ...
- And many more ...

- Real and complex
- Single/double precision
- 3 accuracy modes
  - High accuracy
    - (Almost correctly rounded)
  - Low accuracy
    - (2 lowest bits in error)
  - Enhanced performance
    - (1/2 the bits correct)

Vector-based elementary functions allow developers to balance accuracy with performance

Vector Statistics

Random Number Generators (RNGs)	Psuedo-random, quasi-random, and non-deterministic generators
	Continuous and discrete distributions of various common distribution types
Summary Statistics (SS)	Parallelized algorithms for computation of statistical estimates for raw multi-dimensional datasets.
Convolution/ correlation	A set of routines intended to perform linear convolution and correlation transformations for single and double precision real and complex data.

Intel® MKL Sparse Solvers

PARDISO – Parallel Direct Sparse Solver	Support a wide range of matrix types.
	Based on BLAS level 3 update and pipelining parallelism.
	Supports out-of-core execution for huge problem sizes.
	New: Cluster support.
DSS – Direct Sparse Solver Interface for PARDISO	An alternative, simplified interface to PARDISO.
ISS – Iterative Sparse Solver	Symmetric positive definite: CG solver.
	Non-symmetric indefinite: Flexible generalized minimal residual solver.
	Based on Reverse Communication Interface (RCI).

More Intel® MKL Components

Data Fitting

1D linear, quadratic, cubic, step-wise const, and user-defined splines

Spline based interpolation/extrapolation

PDEs (Partial Differential Equations)

Solving Helmhotz, Poisson, and Laplace problems.

Optimization Solvers

Solvers for nonlinear least square problems with/without constraints

Support Functions

Memory management

Threading control

…

What are Intel MKL DNN Primitives?

A set of performance primitives to speed up image recognition topologies on existing or custom NN frameworks

- Topologies: AlexNet, VGG, GoogleNet, ResNet
- Frameworks: Caffe*, TensorFlow*, CNTK*, Torch*, MXNet*, ...

Operations (forward/backward)	Algorithms
Activation	ReLU
Normalization	batch, local response
Pooling	max, min, average
Convolutional	fully connected, direct batched convolution
Inner product	forward/backward propagation of inner product computation
Data manipulation	layout conversion, split, concat, sum, scale

NN Primitives API Examples

Function	Description
`dnnConvolutionCreateForwardBias_F32(&primitive, attributes, dnnAlgorithmConvolutionDirect, dimension, inputSize, outputSize, filterSize, stride, inputOffset, dnnBorderZeros);`	Create a convolution primitive for forward pass. This only creates a descriptor of the operation. Input and output data is not specified yet.
`dnnExecute(primitive, inputs, outputs)`	Execute the primitive. Input and output data is specified at the execution time.
`dnnLayoutCreate(&layout, params)`	Create a user defined data layout by specifying number of dimensions, and size and stride for each dimension.

`dnnLayoutCreateFromPrimitive(&layout, primitive, type)`
 	
`dnnAllocateBuffer(&ptr, layout)`
	Query the layout required by a primitive.
	
	Allocate memory buffer for converted layout.

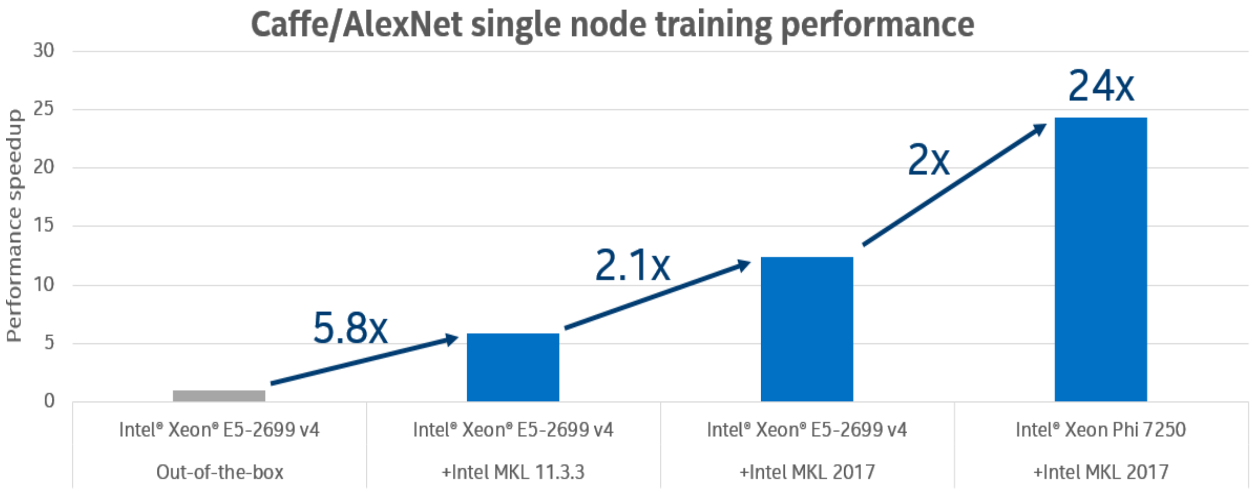
`if (!dnnLayoutCompare(l1, l2))` `    dnnConversionCreate(&conversion_prim, l1, l2)`	Compare different layout types. Create a conversion operation if necessary.

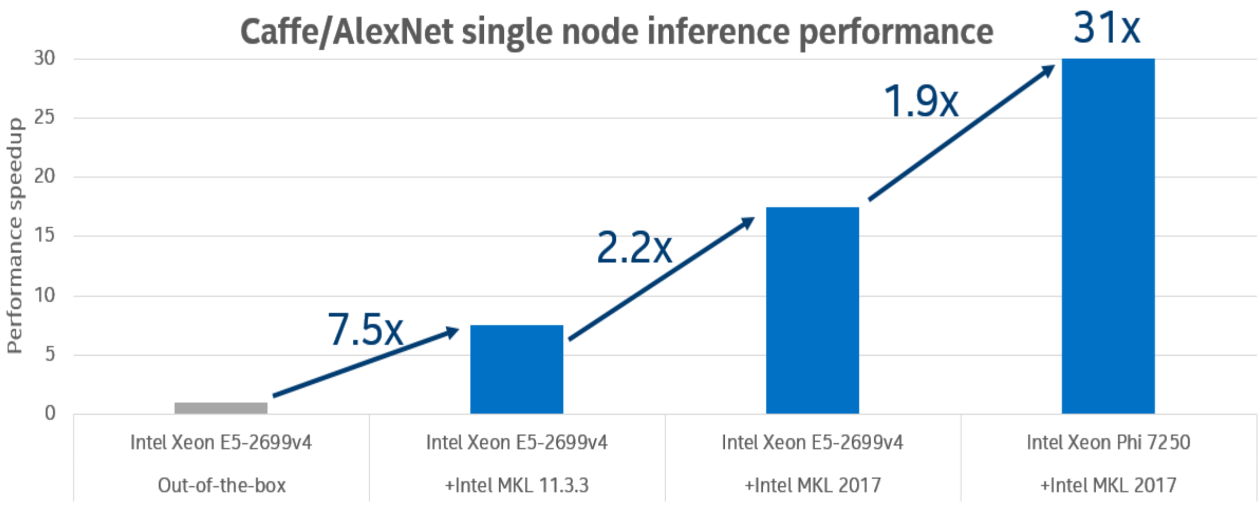
DNN Primitives in Intel® MKL Highlights

A plain C API to be used in the existing DNN frameworks

Brings IA-optimized performance to popular image recognition topologies:

– AlexNet, Visual Geometry Group (VGG), GoogleNet, and ResNet





Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance . *Other names and brands may be property of others

Configurations:

- 2 socket system with Intel® Xeon® Processor E5-2699 v4 (22 Cores, 2.2 GHz,), 128 GB memory, Red Hat* Enterprise Linux 6.7, BVLC Caffe, Intel Optimized Caffe framework, Intel® MKL 11.3.3, Intel® MKL 2017
- Intel® Xeon Phi™ Processor 7250 (68 Cores, 1.4 GHz, 16GB MCDRAM), 128 GB memory, Red Hat* Enterprise Linux 6.7, Intel® Optimized Caffe framework, Intel® MKL 2017

All numbers measured without taking data manipulation into account.

What's New: Intel® MKL 2017

- Optimized math functions to enable neural networks (CNN and DNN) for deep learning
- Improved ScaLAPACK performance for symmetric eigensolvers on HPC clusters
- New data fitting functions based on B-splines and monotonic splines
- Improved optimizations for newer Intel processors, especially Knight's Landing Xeon Phi
- Extended TBB threading layer support for all BLAS level-1 functions

Intel(R) MKL Resources

Intel® MKL website

- https://software.intel.com/en-us/intel-mkl

Intel MKL forum

- https://software.intel.com/en-us/forums/intel-math-kernel-library

Intel® MKL benchmarks

- https://software.intel.com/en-us/intel-mkl/benchmarks#

Intel® MKL link line advisor

- http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor/

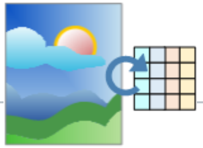
Intel® Data Analytics Acceleration Library
(Intel® DAAL)

Intel® Data Analytics Acceleration Library
(Intel® DAAL)

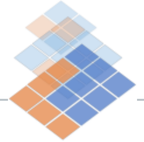
An industry leading Intel® Architecture based data analytics acceleration library of fundamental algorithms covering all machine learning stages.

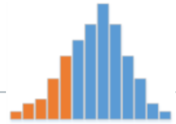
Pre-processing → **Transformation** → **Analysis** → **Modeling** → **Validation** → **Decision Making**

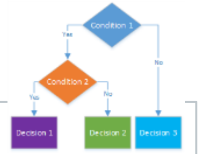
(De-)Compression
Outlier Detection
Normalization



PCA
Statistical moments
Variance matrix
Pp-QR, SVD, Cholesky
Apriori
Sorting







Ridge linear regression
Naïve Bayes
SVM
Classifier boosting

Kmeans
EM GMM

Collaborative filtering

Neural Networks

Intel DAAL Main Features

Building end-to-end data applications

Optimized for Intel architectures, from Intel® Atom™, Intel® Core™, Intel® Xeon®, to Intel® Xeon Phi™

A rich set of widely applicable algorithms for data mining and machine learning

Batch, online, and distributed processing

Data connectors to a variety of data sources and formats: KDB*, MySQL*, HDFS, CSV, and user-defined sources/formats

C++, Java, and Python APIs

PyDAAL (Python API for Intel® DAAL)

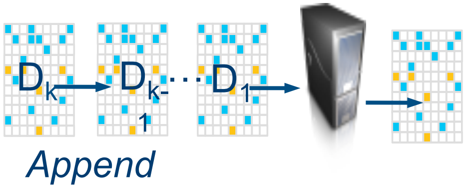
Turbocharged machine learning tool for Python developers

Interoperability and composability with the SciPy ecosystem:

- Work directly with NumPy
- Faster than scikit-learn

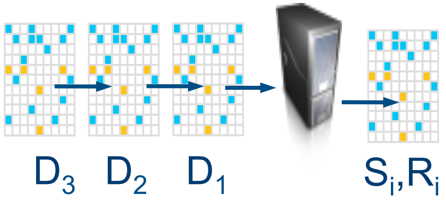
Processing modes

Batch Processing



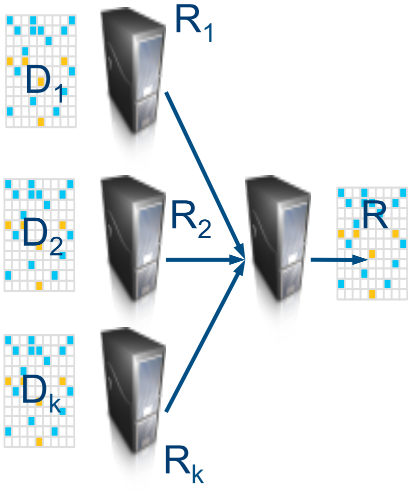
$R = F(D_1,\ldots,D_k)$

Online Processing



$S_{i+1} = T(S_i,D_i)$
$R_{i+1} = F(S_{i+1})$

Distributed Processing



$R = F(R_1,\ldots,R_k)$

	Algorithms	Batch	Distributed	Online
Descriptive statistics	Low order moments	√	√	√
	Quantiles/sorting	√		
Statistical relationships	Correlation / Variance-Covariance	√	√	√
	(Cosine, Correlation) distance matrices	√		
Matrix decomposition	SVD	√	√	√
	Cholesky	√		
	QR	√	√	√
Regression	Linear/ridge regression	√	√	√
Classification	Multinomial Naïve Bayes	√	√	√
	SVM (two-class and multi-class)	√		
	Boosting (Ada, Brown, Logit)	√		
Unsupervised learning	Association rules mining (Apriori)	√		
	Anomaly detection (uni-/multi-variate)	√		
	PCA	√	√	√
	KMeans	√	√	
	EM for GMM	√		
Recommender systems	ALS	√	√	
Deep learning	Fully connected, convolution, normalization, activation layers, model, NN, optimization solvers,	√		

Intel® DAAL Neural Networks Support

General purpose API for building typical NN topologies

Tensors

- Multi-dimensional data structures to represent complex data

Layers

- Forward and backward computation

Topology

- Predefined structure of a neural network

Optimization solver

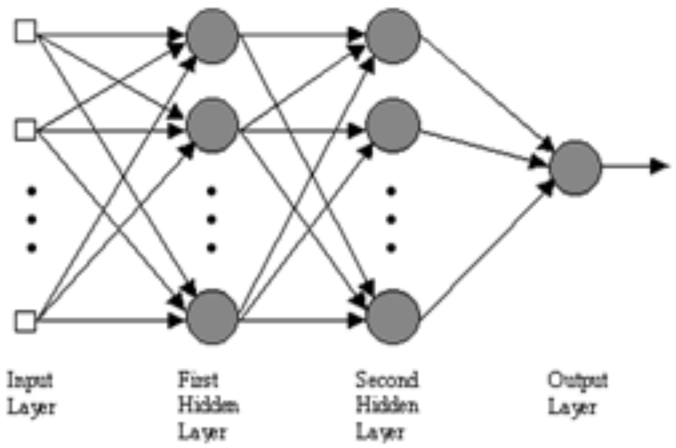
- Computing weights and biases to minimize the objective function

Model

- A network fleshed out with weights and biases of each layer fully defined

Driver

- Engine that drives training and scoring



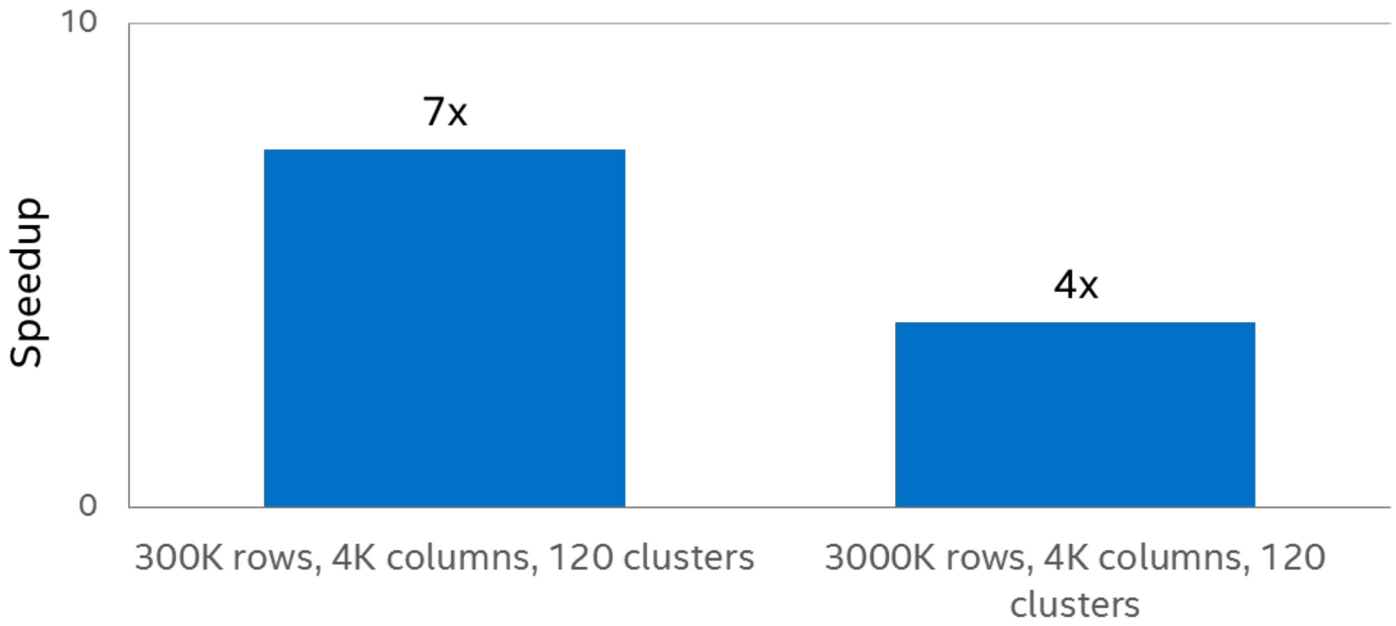
http://www.mu-sigma.com/analytics/thought_leadership/cafe-cerebral-neural-network.html

Compare NN Features in Intel MKL and Intel DAAL

	Intel MKL	Intel DAAL
DNN primitives	Performance critical	Easy integration and high performance
DNN layers	No	All building blocks for NN topology
Optimization solvers	No	Yes
Performance	Top in the class, full control from user side	On-par with Intel MKL
Distributed memory	Not easy, yet	Can be integrated with Spark, MPI cluster, …
Language support	C	C++, Java, Python
Target audience	Users who want to speed up existing frameworks	Users who want to build from scratch or prototype

Intel® DAAL vs. Spark* Mllib

K-means Performance Comparison on Eight-node Cluster

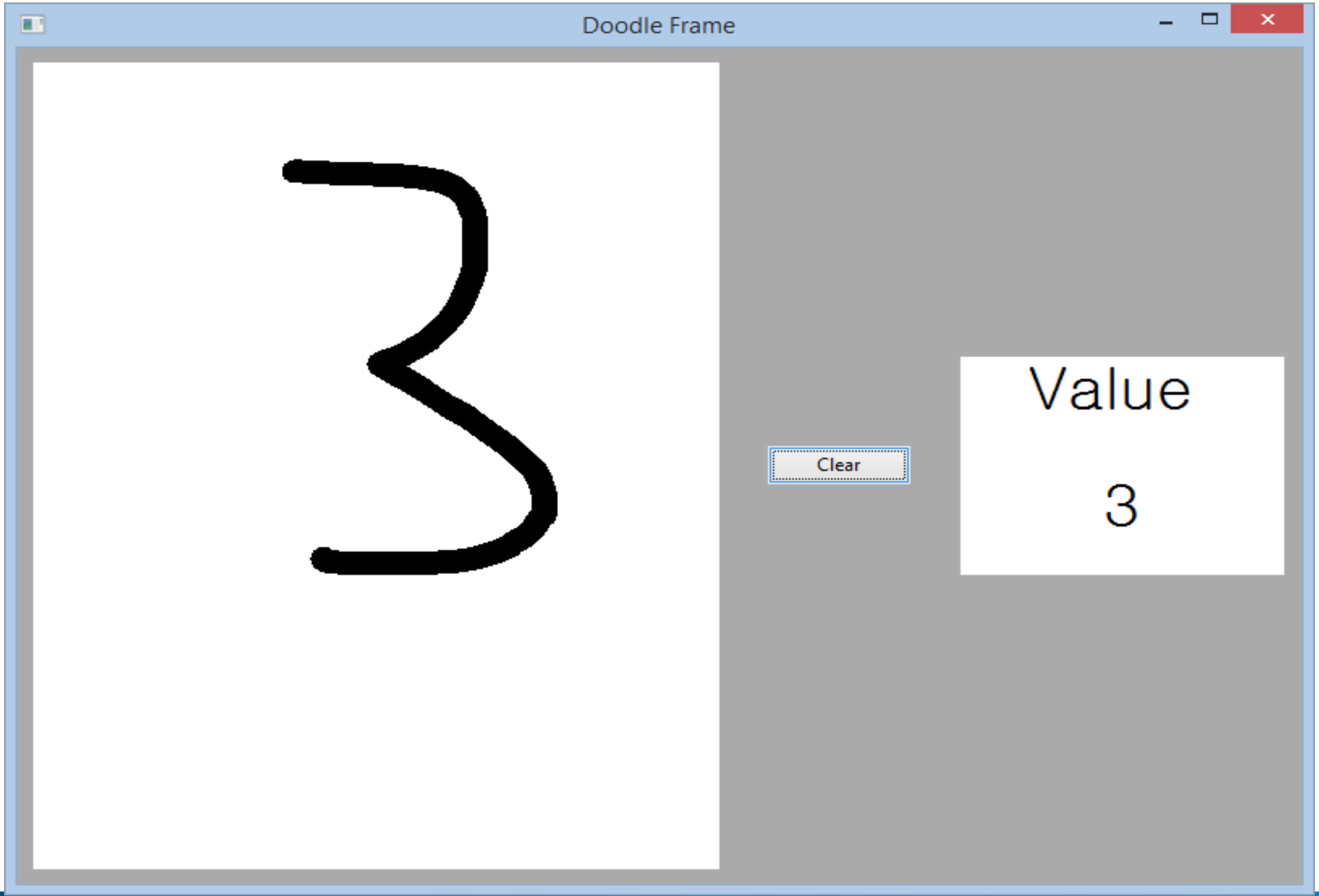


Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2017, Spark 1.2; Hardware: Intel® Xeon® Processor E5-2699 v3, 2 Eighteen-core CPUs (45MB LLC, 2.3GHz), 128GB of RAM per node; Operating System: CentOS 6.6 x86_64.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Demo: Handwritten Digit Recognition



Handwritten Digit Recognition

Training multi-class SVM for 10 digits recognition.

3,823 pre-processed training data.

– available at http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

99.6% accuracy with 1,797 test data from the same data provider.

Confusion matrix:

177.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
0.000	181.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
0.000	2.000	173.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
0.000	0.000	0.000	176.000	0.000	1.000	0.000	0.000	3.000	3.000
0.000	1.000	0.000	0.000	179.000	0.000	0.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000	0.000	180.000	0.000	0.000	0.000	2.000
0.000	0.000	0.000	0.000	0.000	0.000	180.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	170.000	1.000	8.000
0.000	3.000	0.000	0.000	0.000	0.000	0.000	0.000	166.000	5.000
0.000	0.000	0.000	2.000	0.000	1.000	0.000	0.000	2.000	175.000

Average accuracy: 0.996
Error rate: 0.004
Micro precision: 0.978
Micro recall: 0.978
Micro F-score: 0.978
Macro precision: 0.978
Macro recall: 0.978
Macro F-score: 0.978

Training Handwritten Digits

```
void trainModel()
{
        /* Initialize FileDataSource<CSVFeatureManager> to retrieve input data from .csv file */
        FileDataSource<CSVFeatureManager> trainDataSource(trainDatasetFileName,
                DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);

        /* Load data from the data files */
        trainDataSource.loadDataBlock(nTrainObservations);

        /* Create algorithm object for multi-class SVM training */
        multi_class_classifier::training::Batch<> algorithm;

        algorithm.parameter.nClasses = nClasses;
        algorithm.parameter.training = training;

        /* Pass training dataset and dependent values to the algorithm */
        algorithm.input.set(classifier::training::data,trainDataSource.getNumericTable());

        /* Build multi-class SVM model */
        algorithm.compute();

        /* Retrieve algorithm results */
        trainingResult = algorithm.getResult();

        /* Serialize the learned model into a disk file */
        ModelFileWriter writer("./model");
        writer.serializeToFile(trainingResult->get(classifier::training::model));
}
```

Create a numeric table

Create an alg. Obj.

Set input and parameters

Compute

Get result

Serialize the learned model

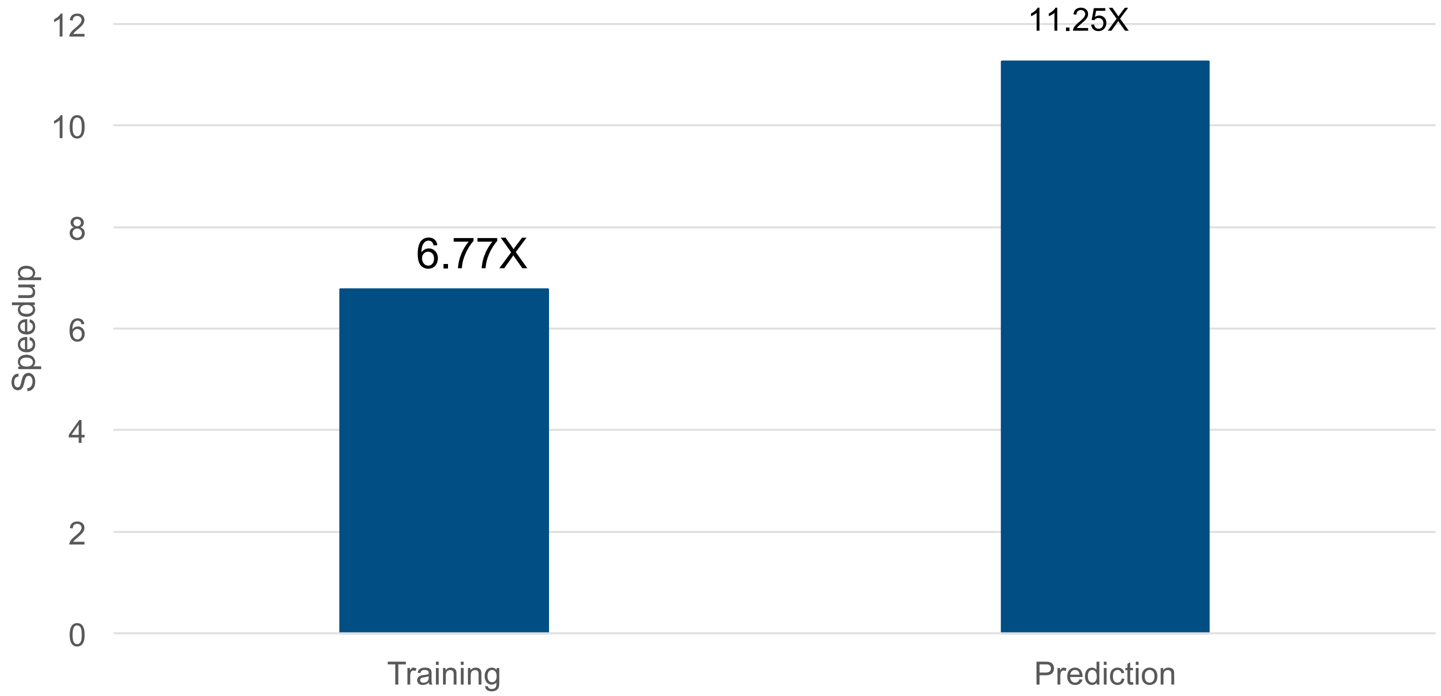
Optimization Notice

Handwritten Digit Prediction

```
void testDigit()
{
        /* Initialize FileDataSource<CSVFeatureManager> to retrieve the test data
from .csv file */
        FileDataSource<CSVFeatureManager> testDataSource(testDatasetFileName,
                DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);
        testDataSource.loadDataBlock(1);

        /* Create algorithm object for prediction of multi-class SVM values */
        multi_class_classifier::prediction::Batch<> algorithm;

        algorithm.parameter.prediction = prediction;

        /* Deserialize a model from a disk file */
        ModelFileReader reader("./model");
        services::SharedPtr<multi_class_classifier::Model> pModel(new
multi_class_classifier::Model());
        reader.deserializeFromFile(pModel);

        /* Pass testing dataset and trained model to the algorithm */
        algorithm.input.set(classifier::prediction::data,
testDataSource.getNumericTable());
        algorithm.input.set(classifier::prediction::model, pModel);

        /* Predict multi-class SVM values */
        algorithm.compute();

        /* Retrieve algorithm results */
        predictionResult = algorithm.getResult();

        /* Retrieve predicted labels */
        predictedLabels = predictionResult->get(classifier::prediction::prediction);
}
```

Deserialize learned model

SVM Performance Boosts Using Intel® DAAL vs. scikit-learn on Intel® CPU



Configuration Info - Versions: Intel® Data Analytics Acceleration Library 2016 U2, scikit-learn 0.16.1; Hardware: Intel Xeon E5-2680 v3 @ 2.50GHz, 24 cores, 30 MB L3 cache per CPU, 256 GB RAM; Operating System: Red Hat Enterprise Linux Server release 6.6, 64-bit.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Intel® DAAL+ Intel® MKL = Complementary Big Data Libraries Solution

Intel MKL	Intel DAAL
C and Fortran API Primitive level	Python, Java & C++ API High-level
Processing of homogeneous data in single or double precision	Processing heterogeneous data (mix of integers and floating point), internal conversions are hidden in the library
Type of intermediate computations is defined by type of input data (in some library domains higher precision can be used)	Type of intermediate computations can be configured independently of the type of input data
Most of MKL supports batch computation mode only	3 computation modes: Batch, streaming and distributed
Cluster functionality uses MPI internally	Developer chooses communication method for distributed computation (e.g. Spark, MPI, etc.) Code samples provided.

Summary

Intel® DAAL is the only data analytics library optimized for current and future Intel® Architectures.

Product page:

- https://software.intel.com/en-us/intel-daal

Forum:

- https://software.intel.com/en-us/forums/intel-data-analytics-acceleration-library

Legal Disclaimer & Optimization Notice